VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Grafická knihovna pro ARM Cortex-M3

Graphics Library for ARM Cortex-M3

2014 Adam Bogocz

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Adam Bogocz**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Grafická knihovna pro ARM Cortex-M3
Graphics Library for ARM Cortex-M3

Zásady pro vypracování:

Cílem bakalářské práce je navrhnout a implementovat knihovnu grafických funkcí pro moduly s procesorem ARM Cortex-M3. Knihovna bude obsahovat základní prvky GUI, prvky pro vizualizaci hodnot v průmyslových řídicích systémech, zobrazení 2D grafů. Knihovna bude navržena modulárně s ohledem na omezenou paměť a výpočetní výkon. Předpokládá se podpora dotykového ovládání.

1. Procesory ARM Cortex a podpora grafického výstupu.
2. Základní vykreslovací metody, podpora dotykového ovládání.
3. Základní prvky GUI – okno, tlačítko, checkbox, radiobutton, atd.
4. Prvky pro vizualizaci průmyslových procesů – ručičkové ukazatele, stupnice, teploměry, atd.
5. Vykreslování 2D grafů – sloupcové, spojnicové a koláčové grafy.

Součástí práce bude sada příkladů demonstrující použití jednotlivých grafických prvků knihovny.

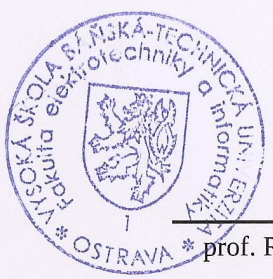
Seznam doporučené odborné literatury:

[1] Joseph Yiu, The Definitive Guide to the ARM Cortex-M3, Newnes; 2 edition, 2009, ISBN 1856179638
[2] Andrew Sloss, Dominic Symes, Chris Wright, ARM System Developer's Guide: Designing and Optimizing System Software, Morgan Kaufmann, 2004, ISBN 1558608745
[3] Peter Shirley, Michael Ashikhmin, Steve Marschner, Fundamentals of Computer Graphics, A K Peters/CRC Press; 3 edition, 2009, ISBN 1568814690

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Ing. Michal Krumnikl**

Datum zadání: 16.11.2012
Datum odevzdání: 07.05.2014



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014 .....

Rád bych na tomto místě poděkoval vedoucímu bakalářské práce Mgr. Ing. Michalu Krumniklovi za jeho podporu, Andymu Brownovi za jeho knihovnu stm32plus a bývalému kolegovi Krzysztofu Dziubovi za jeho pomoc při zprovozňování prvního kompilátoru a demo aplikací.

Abstrakt

Práce si klade za cíl vytvořit nezávislou knihovnu v C++ pro tvorbu uživatelských rozhraní. Knihovna má být co nejméně náročná na systémové prostředky, jako jsou operační nebo programová paměť, protože její cílové nasazení jsou procesory architektury ARM Cortex-M3. Hlavním úkolem bude jednoduchá použitelnost (API) z hlediska uživatele-programátora a nezávislost na dalších knihovnách nebo OS, jako FreeRTOS. Knihovna bude poskytovat základní GUI prvky, jako jsou tlačítka, přepínače nebo zaškrtávací pole.

Klíčová slova: C++, GUI knihovna, ARM, Cortex-M3, API

Abstract

This work aims to create an independent C++ library for creation of user interfaces. The library should be system resources friendly, such as operating or program memory, because its target deployment architecture are ARM Cortex-M3 processors. The main task is simple interface(API) for the user-programmer and independecy from other libraries or OS like FreeRTOS. The library will provide basic GUI elements such as buttons, radio buttons or check boxes.

Keywords: C++, GUI library, ARM, Cortex-M3, API

Seznam použitých zkratek a symbolů

API	–	Application programming interface
GUI	–	Graphical user interface
OS	–	Operating system

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

V dnešní době je použití mikroprocesorů všestranné a velice rozšířené, protože jejich pořizovací cena není vysoká. Praktické nasazení lze vidět všude, začíná to u levnějších zařízení, jako jsou mikrovlnné trouby, pračky, teploměry, a končí u dražších mobilních telefonů, tabletů, herních konzolí, televizí.

Ovšem cílová kategorie produktů této bakalářské práce nejsou drahá zařízení, protože produkty, jako televize či telefony, mají svůj vlastní, většinou proprietární OS, který veškerou funkcionalitu již obsahuje. Na druhou stranu v zařízení, jako je například pračka, mikrovlnná trouba, chytrý termostat, by se dotykový panel s příjemným grafickým rozhraním využil.

Pro prototypování a implementaci jsem použil vývojový kit. Dříve byly vývojové kity drahé, a tudíž pro domácí příležitostný vývoj nevhodné, ale dnes to již naštěstí neplatí. V dnešní době je možné vývojový kit zakoupit za velice příznivou cenu a rozšířit jej o displej nebo jiné senzory či sběrnice.

Vývojářské kity se vyrábí v mnoha variantách, od plně modulárních, kde je každá komponenta samostatný modul, až po rozsáhlejší, kde se v základní sestavě nachází např. tlačítka, externí hodiny, čtečky karet a USB porty. Pro implementaci a praktickou ukázku jsem použil kit MINI-STM32, ve kterém je procesor STM32F103VE od ST Microelectronics.

Tvorba grafických rozhraní však není tak jednoduchým úkolem, protože v dnešní době jsou vývojáři zvyklí na robustní a pohodlné desktopové frameworky, které fungují úplně samy. Ovšem pro platformu ARM už není takový výběr knihoven jako pro desktopy, navíc jsou s existujícími knihovnami spojeny tyto nedostatky :

- nedostupnost zdrojového kódu, tudíž nemožná úprava
- placená licence pro použití
- závislost na dalších knihovnách
- neatraktivní vzhled
- komplikovanost

Jako inspiraci pro tvorbu GUI knihovny jsem si vybral Qt, jelikož jsem ji sám použil pro několik projektů a její API se mi zdá velice srozumitelné a snadno rozšiřitelné. Bohužel jsem se rozhodl upustit od některých architektonických návrhů v Qt, jelikož plánovaná knihovna musí být nenáročná na paměť a to Qt zrovna není.

2 Použitý hardware

Pro splnění cíle bakalářské práce je z hlediska vybavení nutný vývojový kit a programátor/debugger.

2.1 Vývojový kit MINI-STM32

Jedná se o malý, levný vývojový kit (obrázek č.1), obsahující :

- ARM procesor STM32F103VE
- 2",4 TFT LCD modul s dotykovým ovladačem
- CR1220 baterii pro RTC
- MicroSD slot
- RS-232 s DSUB konektorem
- mikrospínač
- LED diodu
- USB port
- JTAG port

2.1.1 STM32F103VE

Jedná se o mikroprocesor rodiny Cortex-M3, založený na architektuře ARMv7-M a navržený pro dosažení vysokého výkonu a zároveň maximální efektivity levných vestavěných aplikací. Takový procesor je vhodné nasadit např. do průmyslových řídících systémů, automobilové elektroniky nebo automatizace domácnosti. Klíčové vlastnosti jsou:

- 32-bitové jádro
- maximální operační frekvence 72 MHz
- 512 kB programové paměti
- 64 kB operační paměti
- 3x12-bit A/D převaděče
- 2x12 bit D/A převaděče
- ladící rozhraní JTAG a SWD
- 2xI2C, 5xUSART, 3xSPI, CAN, USB, SDIO

Obrázek 1: Vývojový kit MINI-STM32

Procesor je schopný pracovat jak v režimech nízké spotřeby, tím pádem nižšího výkonu, tak v režimu vysokého výkonu a vyšší spotřeby. Tyto režimy se nastavují za běhu aplikace, nastavováním zdroje taktovací frekvence a frekvenčními děliči, které následují. V porovnání s procesory ATMEGA, se kterými mám praktické zkušenosti, je nastavovaní taktovací frekvence a děliček mnohem komplikovanější, protože musím nakonfigurovat všechny děliče, které vedou k periferii, navíc se periferie nastavují po skupinách. Obrázek č.2 zobrazuje, jak funguje taktování a kolik je v procesoru děličů.

Podrobnější informace lze nalézt v publikaci [1].

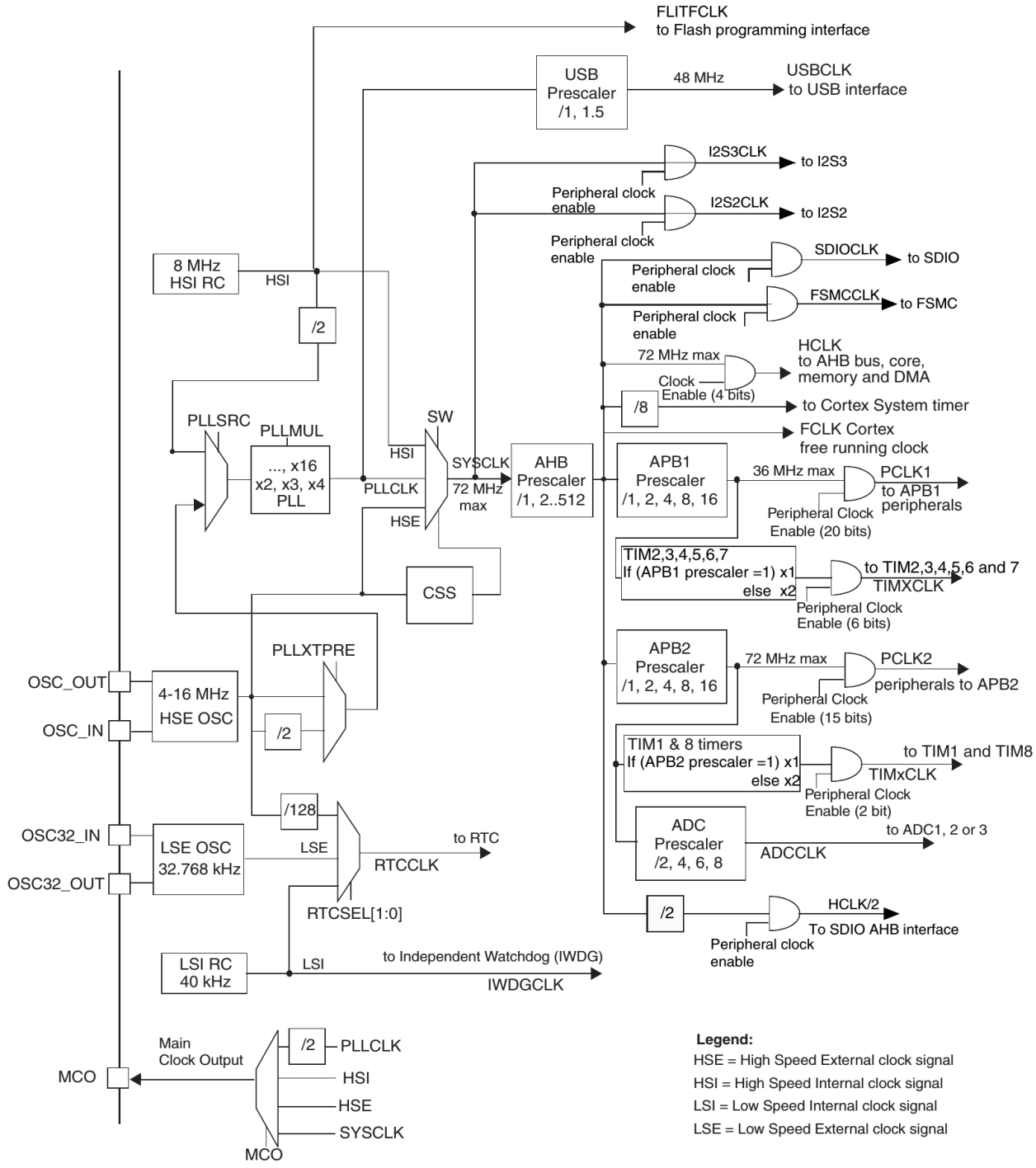
2.1.2 LCD modul

Modul se skládá ze tří komponent:

- 2”,4 TFT LCD panel s rozlišením 240x320
- ovladač displeje ILI9325
- ovladač dotyku TSC2046

Ovladač TFT LCD displejů ILI9325 podporuje 262 144 barev a dá se s ním komunikovat čtyřmi způsoby, a to i80 (s šířkou sběrnice 8/9/16/18 bitů), VSYNC, SPI a RGB (s šířkou sběrnice 6/16/18 bitů) rozhraními. Podporuje také režim úspory energie, a to ve 3 funkcích:

- 8 barev
- pohotovostní režim (STANDBY)



Obrázek 2: Strom frekvenčních děličů v STM32F103VE

- řežim spánku

Procesor a ovladač displeje jsou propojené tak, že k němu přistupuji po rozhraní i80 16-bitů.

Ovladač dotyku TSC2046 je nástupcem ADS7846, se kterým má 100% kompatibilní vývody. Jedná se o multifunkční čip, který kromě měření dotyku, umí měřit i teplotu nebo napětí. Pro komunikaci využívám rozhraní SPI a pin PEN. Při dotyku na LCD se PEN přepne z log. 1 na log. 0, čímž dává najevo, že si mám vyzvednout pomocí SPI souřadnice stisku.

2.2 Segger J-Link ARM

Jedná se o ladicí sondu pro ARM procesory, která je schopná procesor naprogramovat i ladit pomocí JTAG rozhraní, a to až do rychlosti 3 MB/s. Dokáže měřit napájecí napětí testovaného procesoru nebo ladit vícejádrové procesory. K PC se připojuje pomocí USB. Výrobce dodává základní balíček pro Linux, obsahující konzolový GDB server a konzolový jflash, který je schopný aktualizovat firmware v sondě a zapisovat do paměti procesoru. Na druhou stranu balíček dodávaný pro Windows obsahuje širokou škálu aplikací, všechny mají GUI a jsou velice kvalitní.

3 Použitý software

V této části bych rád popsal, jaká vývojová prostředí a knihovny jsem použil nebo vyzkoušel a proč jsem se pro mnou vybrané rozhodl.

3.1 Kompilátory

Jako první jsem zkompiloval sadu GCC ARM ELF, u které jsem narazil na problém zprovoznění C++ a jeho standartní knihovny, takže jsem nainstaloval kompilátor z balíčkovacího systému, což mi zajistilo kompilovatelnost C++ aplikací, ale měl jsem problémy s call stackem a aplikace byly velice nestabilní. Jako nejlepší volba se ukázal CodeSourcery G++ Lite, který byl zkompilován jako arm-none-eabi, a fungoval výborně. Bohužel plná verze je zpoplatněna, ale pro mé potřeby stačila ta základní.

Kompilace už nebyla problém, ovšem linkování zkompilovaných objektů není tak přímočaré jako u PC, protože se linkeru musí poskytnout linkovací skript. Jedná se o soubor, který specifikuje, v jakém paměťovém rozsahu se nachází paměť programu, RAM, jak velký má být call stack a spoustu dalších parametrů. Jeden takový skript jsem napsal a fungoval, byl ale velice primitivní. Naštěstí knihovny nebo vývojová prostředí v sobě mají připravené skripty na míru pro konkrétní procesory.

3.2 Vývojová prostředí

Jedno z nejznámějších vývojových prostředí se jmenuje Keil a je vyvíjený přímo od ARM, je bohužel licencovaný a pouze pro platformu Windows. Veškerá dema a příklady, které jsem stáhl od výrobce, byly připravené právě pro Keil. Dokonce návod od výrobce ladící sondy popisoval konfiguraci pod Keilem. Prostředí obsahuje kompilátor a je dobře konfigurovatelné. Stažené příklady v něm ihned fungovaly a šly jednoduše ladit v procesoru díky integraci sondy.

Jako další jsem zkusil Eclipse s rozšířením GNU ARM Eclipse. Ten už sice neobsahuje tak automatizovanou konfiguraci, ale umí pracovat s různými kompilátory naprosto bez problémů. Dobře funguje jak pod Windows, tak pod Linuxem. Ladění jsem zprovoznil pomocí nastartování J-Link GDB serveru a v Eclipse nakonfiguroval profil pro vzdálené ladění po TCP/IP. GDB server zvládne dokonce i naprogramovat procesor, takže není třeba programování řešit separátně. Co se rychlosti a pohodlí týče, je Keil výborný, avšak kvůli licencování a preferování platformy Linux jsem dále při práci na ARM používal Eclipse. Pro práci na kódu kompilovatelném na PC jsem používal multiplatformní Qt Creator z Qt SDK pro jeho rychlost a jednoduchost.

3.3 Knihovny

Pro snazší práci s procesorem jsem našel velice dobře zpracovanou knihovnu stm32plus od Andyho Browna, která obsahuje upravené zdrojové kódy standartní knihovny pro C++ a spoustu pohodlných funkcí pro nastavování a práci s periferiemi. Kromě linkovacího skriptu také obsahuje implementaci přístupu do mého ovladače displeje a dotyků

a řeší přerušení. Díky této knihovně jsem se nemusel soustředit na detaily komunikace mezi procesorem a ovladačem displeje. Knihovna se kompiluje pomocí SConstruct nebo ji lze naimportovat do Eclipse, kde funguje okamžitě.

Jako demonstrace použití knihovny slouží ukázka 1

```
typedef Fsmc16BitAccessMode<FsmcBank1NorSram1> LcdAccessMode;
typedef ILI9325_Portrait_262K<LcdAccessMode> LcdPanel;

LcdAccessMode *_accessMode;
LcdPanel *_gl;

GpioE<DefaultDigitalOutputFeature<1> > pe;
GpioD<DefaultFsmcAlternateFunctionFeature<11>> pd;
Fsmc8080LcdTiming fsmcTiming(0,2);

_accessMode=new LcdAccessMode(fsmcTiming,16,pe[1]);
_gl=new LcdPanel(*_accessMode);

ILI9325Gamma gamma(0x0006,0x0101,0x0003,0x0106,0x0b02,0x0302,0x0707,0x0007,0x0600,0
    x020b);
_gl->applyGamma(gamma);

_gl->setBackground(0);
_gl->clearScreen();

Point p1,p2;
int i;

prompt("Line test");

for(i=0;i<5000;i++) {
  p1.X=rand() % _gl->getXmax();
  p1.Y=rand() % _gl->getYmax();
  p2.X=rand() % _gl->getXmax();
  p2.Y=rand() % _gl->getYmax();

  _gl->setForeground(rand());
  _gl->drawLine(p1,p2);
}
```

Výpis 1: Demo pro náhodné kreslení čar

Ovladač pro displej fungoval bez problémů, bohužel ovladač na čtení dotyků nefungoval. Problém jsem nakonec lokalizoval v knihovně a opravil jej, způsobilo ho nestandartní zapojení uvnitř vývojové sady.

4 Grafická knihovna

V této části se budu věnovat popisu toho, jaké existující knihovny jsem studoval, jak jsem se jimi inspiroval k využití principů nebo naopak k rozpoznání toho, co by můj projekt obsahovat neměl. Nakonec popíšu implementaci celé knihovny.

4.1 Výzkum

Jako programátor-uživatel mám zkušenosti s několika grafickými knihovnami pro PC. Vzhledem k tomu, že se hlavní parametry jakýchkoliv grafických knihoven dají porovnávat mezi sebou, neměl bych mít při srovnávání s ARM knihovnami problém.

S jako úplně první knihovnou bežící pod ARM jsem se setkal s μC/GUI od Micrium. Obsahuje širokou škálu prvků, včetně více okenních rozhraní, designem silně připomíná Windows 3.11 viz. obrázek 3 a je závislé na RTOS (nebo lze dopsat vlastní OS). Mezi asi největší nevýhodu tohoto řešení patří placená komerční licence, z tohoto důvodu jsem bohužel nemohl porovnat, jak se knihovna používá. Na druhou stranu má v sobě zabudovaný designér GUI, ve kterém je možné navrhovat, aniž by uživatel znal C, alespoň podle Micriumu.

Další knihovnou je GUI knihovna od ARM, která je součástí MDK-Professional od verze 4.23, a je tedy integrovaná přímo v rozhraní Keil 3.2. Stejně jako předchozí knihovna je opět placená, má v sobě navíc už zabudovanou podporu pro témata a obsahuje také ovladače pro mnoho LCD řadičů. Bohužel se mi nepodařilo získat ani obrázek dema pro porovnání.

V poslední řadě bych rád řekl něco o grafické knihovně Qt. Jedná se open source knihovnu primárně pro stolní počítače, avšak je již oficiálně podporovaná i na mobilních zařízeních, jako jsou iOS, Android nebo Symbian, což jsou vlastně taky ARM zařízení. Ovšem Qt staví nad hostovaným OS a je navrženo pro systémy s většími paměťovými a výkonovými prostředky. Nicméně jsem s jeho API už nějakou dobu pracoval a zjistil jsem, že mi velice vyhovuje svojí jednoduchostí a přehlednou architekturou. Bohužel není možné použít celý návrh pro realizaci mojí práce, protože Qt není jen grafická knihovna, ale kompletní platforma, která C++ nadstavuje o systémy meta-objektů, SIGNÁL/SLOTů a spoustu dalšího.

4.2 Vize

Na základě výzkumu je mým cílem vytvoření otevřené knihovny s jednoduchým rozhraním, absolutně nezávislé na OS či dalších knihovnách, kromě standartní C++ knihovny. Základem zobrazitelného objektu bude Widget, který může být, stejně jako v Qt, oknem nebo bázovou třídou pro všechny další prvky, jako jsou tlačítka, zaškrtávací pole apod. Každý Widget může obsahovat libovolný počet Widgetů uvnitř, jednoduše pomocí vazby rodič-potomek. Prvek musí být schopný vykreslit pouze svou část a není schopen kreslit mimo svou oblast. Pozicování je vždy relativní vůči rodiči.

Čas a pořadí vykreslování neovlivní uživatel, ale je v kompetenci okenního správce rozhodnout, kdy a kdo bude vykreslen. Tento koncept umožní optimalizaci vykreslo-

Obrázek 3: Ukázka μc/GUI

vací fronty. Okenní správce bude také zodpovědný za doručení dotekových událostí do správného Widgetu. Klíčové komponenty, jako správce obrazovky, nebudou přístupné přímo, ale pomocí systému událostí.

Celá knihovna musí být minimálně náročná na paměť operační i programovou. Součástí bude také simulátor pro demonstraci a rychlý vývoj GUI přímo na PC, který bude 1:1 představovat chování a vzhled na cílovém zařízení.

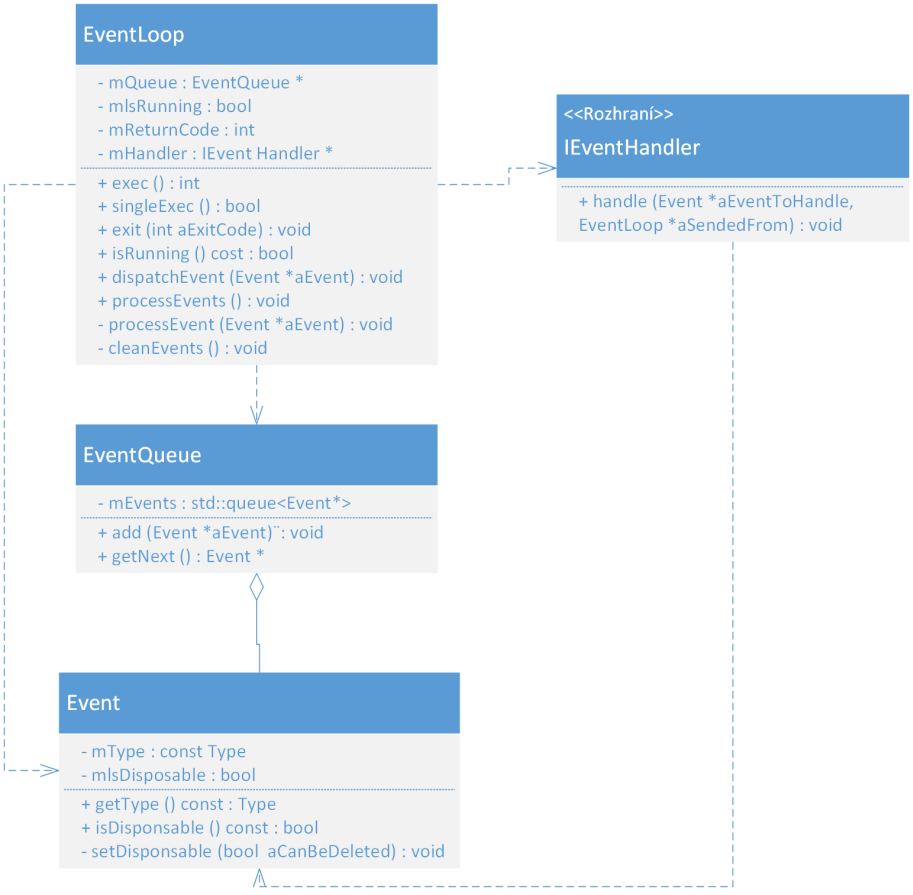
4.3 Implementace

4.3.1 Aplikační smyčka

Pro prvotní inicializaci slouží třída Application, která vytvoří hlavní událostní smyčku a všechny ostatní komponenty potřebné pro funkci knihovny. Třída Application vlastní hlavní událostní smyčku a poskytuje rozhraní pro práci s ní.

Samotná smyčka je implementována třídou EventLoop, která nad kontejnerem EventQueue poskytuje aplikační logiku. Třídu znázorňuje diagram 4 . EventLoop umí přidávat události do fronty, blokovaně i neblokovaně spustit smyčku, zpracovat celou frontu na požádání v blokovaném volání, hlásit svůj stav a dovolí zpětně zjistit ukončovací hodnotu.

V ukázce č.2 je zobrazena implementace klíčových metod. Metody exec() a singleExec() slouží ke spouštění událostní smyčky, kde exec() je blokující volání a singleExec() umožní jednorázové zpracování fronty a zároveň konzistenci stavu aplikace. Třída Application je zveřejňuje jako run() a runOnce(). Ani jednu ze dvou zmíněných metod není možné spustit, když smyčka právě pracuje. Metoda exec() skončí při obdržení požadavku pro ukončení aplikace a zároveň smaže všechny zbylé události ve frontě a vrátí návratovou hodnotu programu, která může reprezentovat ukončení aplikace z důvodu kritické chyby některé z komponent. Při návrhu simulátoru jsem zjistil, že umožnění neblokovaného spuštění fronty je naprosto nezbytné pro uvolnění hlavního vlákna pro platformu, která bude poskytovat plátno pro kreslení nebo vstupy z dotyku či klávesnice.



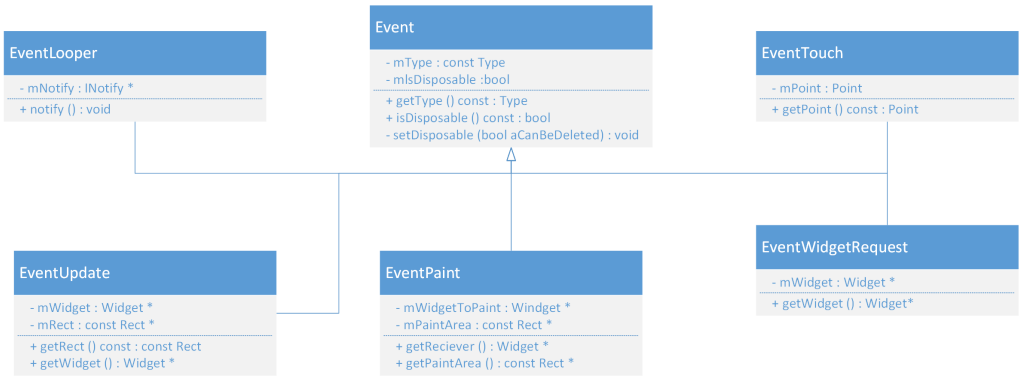
Obrázek 4: UML Diagram smyčky událostí

O přípravu na zpracování události se postará metoda processEvent(), která má na starosti životní cyklus události a předá jí implementaci rozhraní IEventHandler, která řeší jak individuální události zpracovat. O životní cyklus události Event je postaráno právě v EventLoop. Libovolnou implementací třídy Event lze ale tuto funkci vypnout a řídit životní cyklus instancí nové implementace podle potřeby.

```
int EventLoop::exec()
{
    if (mIsRunning)
        return -2;

    mIsRunning = true;
    while(mIsRunning)
    {
        processEvent(mQueue->getNext());
    }

    cleanEvents();

    return mReturnCode;
}

bool EventLoop::singleExec()
{
    if (mIsRunning)
        return false;

    mIsRunning = true;

    Event *e;
    while((e=mQueue->getNext()) != 0)
        processEvent(e);
    bool shoudBeRunningAgain = mIsRunning;
    mIsRunning = false;
    return shoudBeRunningAgain;
}

void EventLoop::processEvent(Event *event)
{
    if (event == 0)
        return;

    event->setDisposable(true);
    mHandler->handle(event, this);

    if (event->isDisposable())
        delete event;
}
```

Výpis 2: Implementace událostní smyčky EventLoop

EventQueue je velice jednoduchý kontejner pro události třídy Event, který umí pouze přidat ukazatel na událost do fronty nebo vrátit ukazatel na první událost ve frontě. Po-



Obrázek 5: Třídní diagram pro třídu Event

kud žádná událost ve frontě není, tak vrátí nulový ukazatel. Pro samotné uložení fronty ukazatelů na třídy Event je použitý std::queue ze standartní knihovny C++.

Pro přidání konkrétní události do fronty je třeba pomocí makra App získat instanci Application a zavolat metodu dispatchEvent(), jak je znázorněno v ukázce č.3

```
void Widget::update()
{
    if ( isVisible () )
        App->dispatchEvent(new EventUpdate(this,getRect()));
}
```

Výpis 3: Příklad odeslání události (úryvek z třídy Widget)

4.3.2 Události

Události jsou reprezentovány třídou Event, která si pamatuje enumerátor Type, určující její konkrétní typ, viz. tabulka č.1. Informaci o typu je možné použít samostatně nebo k přetypování na specializovanou třídu, držící další potřebná data k jejímu zpracování.

Třídní diagram č.5 ukazuje všechny události, které knihovna používá. V následující části se jim jednotlivě budu věnovat.

EventTouch je událost typu E_Type_TouchPress nebo E_Type_TouchRelease a slouží k upozornění správce obrazovky o tom, že proběhl dotyk na obrazovce nebo např. kliknutí myší. Je-li typ E_Type_TouchPress, jedná se o stisk, při uvolnění stisku je typ E_Type_TouchRelease. Třída má v sobě navíc informaci o pozici stisku.

EventPaint je žádost o vykreslení určité komponenty nebo její části. Jedná se o ryze interní třídu, se kterou se programátor/uživatel nedostane do styku, protože tento požadavek zasílá správce obrazovky do správce kreslení.

Události EventWidgetRequest a EventUpdate jsou stejně jako v předchozím případě interní a použité v bázové třídě komponent Widget. Jako informaci o změně stavu/nutnosti se překreslit, vyšle komponenta událost EventUpdate do správce obrazovky, EventWidgetRequest slouží pro přepínání obrazovek.

Typ	Třída
E_Type_TouchPress	EventTouch
E_Type_TouchRelease	EventTouch
E_Type_Paint	EventPaint
E_Type_Update	EventUpdate
E_Type_ShowWidget	EventWidgetRequest
E_Type_CloseWidget	EventWidgetRequest
E_Type_Looper	EventLooper
E_Type_Quit	žádná

Tabulka 1: Tabulka asociace typu události na třídu

EventLooper je speciální druh události, která se při zpracovávání znovu přidá do fronty událostí. Zpracovává se tak, že zavolá onNotify() na instanci rozhraní INotify, kterou si drží. Je vhodné ji použít jako náhradu za časovač, která je synchronní s hlavní smyčkou. Knihovna ji využívá například v TouchManager třídě pro kontrolu stavu dotyku, jehož stav je určovaný ARM přerušením.

4.3.3 Správce obrazovky

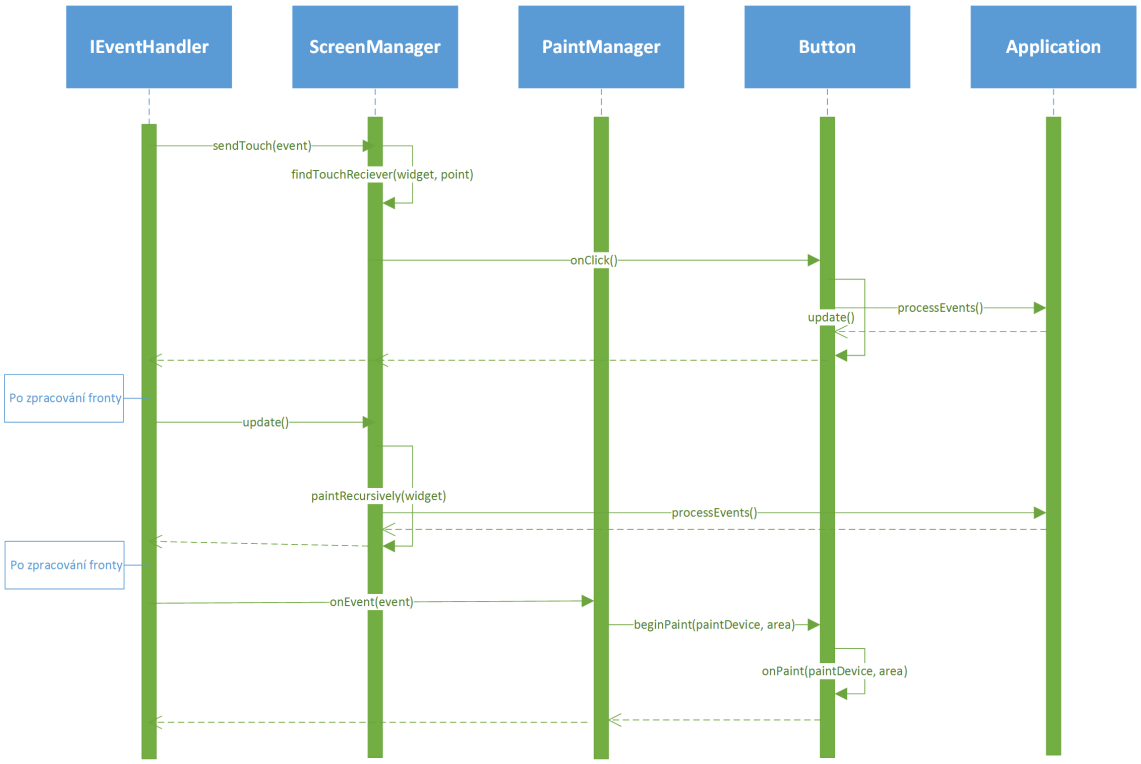
Pro správu oken slouží třída ScreenManager, jenž má za úkol zpracovávat hned několik druhů událostí

- zpracování událostí doteků
- zpracování požadavků na překreslení
- požadavky pro zobrazení a zavření obrazovky

Obdrží-li ScreenManager požadavek na zobrazení obrazovky, ověří, zda-li je nějaká obrazovka aktuálně zobrazena. Pokud je, připraví její schování na pozadí. Následně přidá novou obrazovku na zásobník a připraví rekurzivní kreslení všech komponent, které obrazovka obsahuje. Příprava vykreslování spočívá ve správném seřazení požadavků na kreslení, které obdrží vykreslovací správce.

Zpracování požadavku na překreslení je implementováno tak, že správce rekurzivně projde všechny potomky komponenty a vyšle požadavky na jejich vykreslení. Jedná se o zjednodušenější implementaci než jsem původně plánoval, protože správce měl skládat frontu pro vykreslení, kterou bude postupně optimalizovat na základě dalších překreslovacích požadavků a po určité prodlevě vygeneruje vykreslovací požadavky. Tento proces popisuje sekvenční diagram na obrázku č.6.

Správce při přijetí události EventTouch rozpoznává stavy dotyku při stisku a uvolnění. Pokud správce přijme stisk, najde si komponentu, která se nachází v místě stisku, upozorní ji a uloží si na ni ukazatel. Jakmile správce přijme uvolnění, ověří si, zda-li došlo k uvolnění v oblasti stisklé komponenty. Pokud ano, obdrží komponenta notifikaci o kliknutí a uvolnění, pokud ne, tak pouze o uvolnění.



Obrázek 6: Sekvenční diagram od stisku tlačítka po jeho vykreslení

Obrázek 7: Komponenta Button, vlevo normální stav, vpravo stisk

4.3.4 Komponenta

Nejdůležitější třídou pro uživatele/programátora je Widget. Jedná se o bázovou třídu pro každý zobrazitelný objekt, avšak dá se použít přímo. Z této třídy dědí všechny vestavěné komponenty, které je možné vidět na obrázku č.8

- BarMeter - sloupcový graf
- Button - tlačítko
- ColumnPlot - sloupcový graf
- CheckBox - zaškrtávací pole
- JoinPlot - spojnicový graf
- PiePlot - koláčový graf
- PointerIndicator - ručičkový ukazatel
- RadioButton - rádiové tlačítko
- TextBox - vstupní pole

Komponenty se skládají pomocí konstrukce rodič-potomek, kde se rodič předá jako parametr při konstrukci nebo jej lze metodou setParent() nastavit. Pozicování je vždy relativní vůči svému rodiči, až na výjimku Widgetu bez rodiče, který reprezentuje obrazovku. Rozložení komponent je potřeba ručně dopředu promyslet, protože pozicování není dynamické. Pozici je možné nastavit metodami setPosition() a velikost setDimension(). Některé Widgety nepodporují změnu rozměrů, například u Button nebo CheckBox nelze změnit výšku, pouze šířku. Knihovnu je možné zkompilovat v režimu DEBUG, kdy je každý Widget obkreslený rámečkem, takže je velmi snadné vyladit rozložení.

Vykreslování Widgetů neprobíhá přímo, ale je nutné si o možnost kreslení zažádat, a to z důvodu, aby programátor/uživatel nemohl brzdit aplikační logiku vykreslováním a správce mohl připravit vykreslování ve správném pořadí nebo rozhodnout, zda bude překreslení vůbec potřeba. Zažádat o překreslení může metodou update(), pomocí které je schopný specifikovat i region, který je třeba překreslit. Jakmile je vše připraveno pro vykreslování, správce vykreslování upozorní Widget a předá mu do onPaint() implementaci rozhraní IPaintDevice, pomocí kterého může kreslit tvary a psát text.

Jako názorný příklad slouží výpis č.4, jehož výstup bude vypadat podle obrázku č.7. Před samotným vykreslováním se nastaví posun takový, že každý Widget kreslí od pozice 0,0, takže není třeba řešit, kde začít kreslit. Při implementaci vlastní komponenty je

Virtuální metoda	Popis
onTouch()	Stisku a uvolnění
onClick()	Kliknutí, tzn. stisk a uvolnění na jednom Widgetu
onPaint()	Možnost vykreslit Widget
onHide()	Před schováním obrazovky, pro pozastavení aktivit
onClose()	Před uzavřením obrazovky

Tabulka 2: Tabulka virtuálních metod třídy Widget pro přijetí událostí

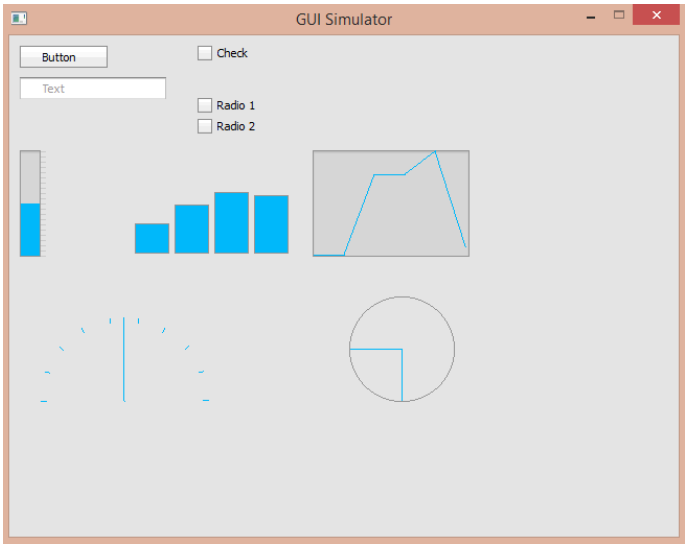
vhodné brát také ohled na parametr oblasti pro překreslení, a to obzvláště u rozměrných Widgetů, obsahujících např. obrázek na pozadí, protože knihovna programátorovi sama nezabrání kreslit mimo požadovanou oblast.

```
void Button::onPaint(IPaintDevice *device, const Rect *)
{
    const Color borderColor = mPushed ? Color::fromRGB(77,80,170) :Color::fromRGB
        (139,139,139);
    const Color * fillColor  = mPushed ? mColorsPushed : mColorsNormal;

    Rect r(Point(),getRect()->getWidth(),getRect()->getHeight());
    device->setColor(&borderColor);
    device->drawRect(&r);

    for(pix line = 1; line < getRect()->getHeight() - 1; line++)
    {
        device->setColor(&fillColor[line-1]);
        device->drawLine(Point(1,line),Point(getRect()->getWidth()-2,line));
    }

    Color font = Color::fromRGB(0,0,0);
    device->setColor(&font);
    device->drawText(Point(22,4),mLabel);
}
```

Výpis 4: Ukázka vykreslování komponenty Button

Pro barevné přechody používám pole instancí třídy Color, ve kterém je každý řádek výplně reprezentován jednou barvou v poli. Třída Color má v sobě předdefinované základní barvy, jako je červená, zelená apod. Dále je z výpisu patrné, že barvy lze vytvářet pohodlně pomocí složek RGB. Pro znázornění oblasti(obdélníku) slouží třída Rect, kterou umí IPaintDevice přímo vykreslit jako prázdný rámeček nebo rámeček s výplní. Tlačítko se kreslí tak, že se nakreslí obvodový rámeček a do něj se vykreslí řádek po řádku postupně čáry. Barva pro každý řádek se použije z pole barev. Nakonec se do tlačítka vypíše text s popisem.

Metoda onPaint() je jen jedna z dalších, pomocí které je Widget upozorněný na události, tabulka č.2 všechny popisuje



Obrázek 8: Všechny implementované komponenty pohromadě

4.3.5 Platformě závislá část

Do této doby byly všechny popsané třídy nezávislé na hardware nebo operačním systému. Pro nasazení na konkrétní platformu/systém je nutné naimplementovat rozhraní IPaintDevice pro kreslení na obrazovku a službu, která bude zasílat události o dotyku na obrazovce.

Pro platformu ARM a knihovnu stm32plus jsem napsal třídy TouchManager a PaintDevice (soubor paintdevice_stm32plus.h). Instance TouchManageru se vytvoří v PaintDevice. TouchManager je třída, která připraví spojení s TC2046 a zajistí vytvoření a odeslání dotekových událostí. Manager získává informace o dotyku tak, že má zaregistrované přerušení z GPIO pinu, které pošle TC2046 ve chvíli doteku, a zaregistrovaný EventLooper. Dotek se rozpozná tím, že přerušení z GPIO nastaví příznak, který se ověřuje při události EventLooper. Je-li příznak pravdivý, započne čtení polohy dotyku po SPI a vytvoří se událost o stisku obrazovky. Detail této implementace je vidět ve výpisu č.5

```
void TouchManager::onNotify()
{
    if (mTouched)
    {
        bool currentlyTouched = _touchScreen->isTouched();
        if (currentlyTouched && mSended)
        {
            return;
        }

        mTouched = currentlyTouched;
        mSended = mTouched;
        _touchScreen->getCoordinates(mTouchPoint);
```

```
            App->dispatchEvent(new EventTouch(Point(mTouchPoint.X,mTouchPoint.Y),
                currentlyTouched));
        }
    }
```

Výpis 5: Ukázka implementace detekce doteku pomocí stm32plus

Kreslící PaintDevice je velice podobný výpisu z ukázkového kódu č.1. Navíc obsahuje převody mezi touto knihovnou a stm32plus. Knihovna neobsahuje obecné algoritmy pro kreslení čar nebo křivek proto, aby se různé operace daly implementovat individuálně a optimalizovat podle platformy, na které se spouští.

Jelikož je součástí knihovny i simulátor pro stolní počítač, bylo nutné naimplementovat IPaintDevice v Qt spolu s rozpoznáváním dotyků (kliknutí myši). Ovšem oproti ARM verzi jsem musel vytvořit třídu QtRunner, která zajistí běh smyčky událostí pro Qt a zároveň umožní zpracování fronty této knihovny.

5 Použití knihovny

Knihovnu je možné zkompilovat předem a staticky nalinkovat nebo ji použít jako projekt a doplnit o vlastní kód, tak jak to demonstruji v ukázkách kódu.

5.1 První aplikace

Způsob využití popíšu na ukázce kódu č.6

```
#include "application.h"
#include "widgets/widget.h"
#include "widgets/button.h"
#include "widgets/checkbox.h"
#include "widgets/radiobutton.h"
#include "radiogroup.h"
#include "showsuperscreen.h"

int main(int argc, char* argv[])
{
    Application a(240,320);

    Widget p;
    Button b("Create",&p);
    b.setPosition(10,10);
    b.setOnClick(new ShowSuperScreen());

    CheckBox f(&p);
    f.setPosition(180,10);

    RadioGroup gr;

    RadioButton rad1("1",&gr,&p);
    rad1.setPosition(180,60);
    RadioButton rad2("2",&gr,&p);
    rad2.setPosition(180,80);

    p.show();

    return a.run();
}
```

Výpis 6: Ukázková aplikace používající ARM knihovnu

Řádkem Application a(240,320) se připraví vše nutné pro běh grafické knihovny a manažerům se nastaví rozlišení obrazovky 240x320 pixelů. Widget p slouží jako hlavní obrazovka, a proto u něj není specifikovaný žádný rodič nebo nastavování pozice. Dále se vytvářejí komponenty Button b, Checkbox f, RadioButtony rad1 a rad2. Jak je z výše uvedeného výpisu zřejmé, nastavil jsem všem pozici a rodiče komponent p, což znamená, že se budou vykreslovat na obrazovku p. Rád bych upozornil na to, že je naprosto validní nastavit např. objektu f rodiče b. Objekt b si pomocí setOnClick() nastaví akci pro stisk tlačítka a tady je zřejmý první rozdíl v použitelnosti oproti Qt, kde jsou veškeré změny a

reakce na ně řešeny systémem SIGNAL/SLOT. Pro objekty rad1 a rad2 bylo nutné vytvořit instanci pomocné třídy RadioGroup, pomocí níž se dá určit, která skupina rádiových tlačítek k sobě patří.

V poslední řadě je nutné specifikovat, která komponenta se má vykreslit, v tomto příkladu Widget p, a také spustit aplikační smyčku pomocí metody exec(). V této chvíli se začnou zpracovávat všechny nahromaděné události a komponenty se vykreslí. Jedná se o velice jednoduchou ukázku a pro reálné nasazení by bylo vhodnější dědit nové třídy z Widget pro každou obrazovku zvlášť, což udrží kód strukturovaný a umožní řešit případy, kdy bude obrazovka překryta druhou a dostane se na pozadí.

Popsaný příklad byl pro ARM platformu, nyní předvedu stejný příklad, ale upravený pro simulaci ve výpisu. Pro zkrácení výpisu jsem kód odlehčil o většinu redundantních řádků. Z výpisu č.7 je patrné, že pro funkčnost simulace, je nutné přidat třídu QtRunner, která v sobě zabaluje vše potřebné pro spouštění Qt a GUI smyčky. Je však obzvlášť důležité, aby došlo ke spuštění hlavní smyčky pomocí QtRunner a ne Application, protože to by vedlo k nefunkčnímu výsledku.

```
// predchozi include
#include "qtrunner.h"

int main(int argc, char* argv[])
{
    Application a(240,320);

    Widget p;
    // vytvareni komponent, ktere se maji zobrazit

    p.show();

    QtRunner r(argc,argv);
    return r.exec();
}
```

Výpis 7: Zjednodušený úryvek kódu pro simulaci

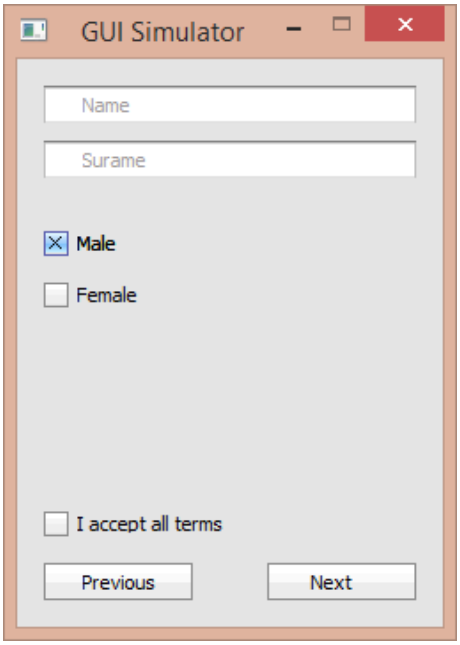
5.2 Kompilace

Pro kompilaci verze ARM je vhodné aplikaci kompilovat v IDE Eclipse s pluginem GNU ARM Eclipse a kompilátorem CodeBench Sourcery G++ Lite. Nejsnadnější je použít přiloženou ukázkovou aplikaci a tu si naimportovat do workspace. Jedná se o konfiguraci od Andyho Browna pro jeho knihovnu stm32plus.

Kompilace verze pro simulaci je mnohem snadnější, stačí pro ni v Qt Creatoru spustit projektový soubor a kompilace bude fungovat okamžitě. Samotný Qt Creator se dá obejít spuštěním příkazu qmake ve složce projektu, který vygeneruje standardní Makefile.

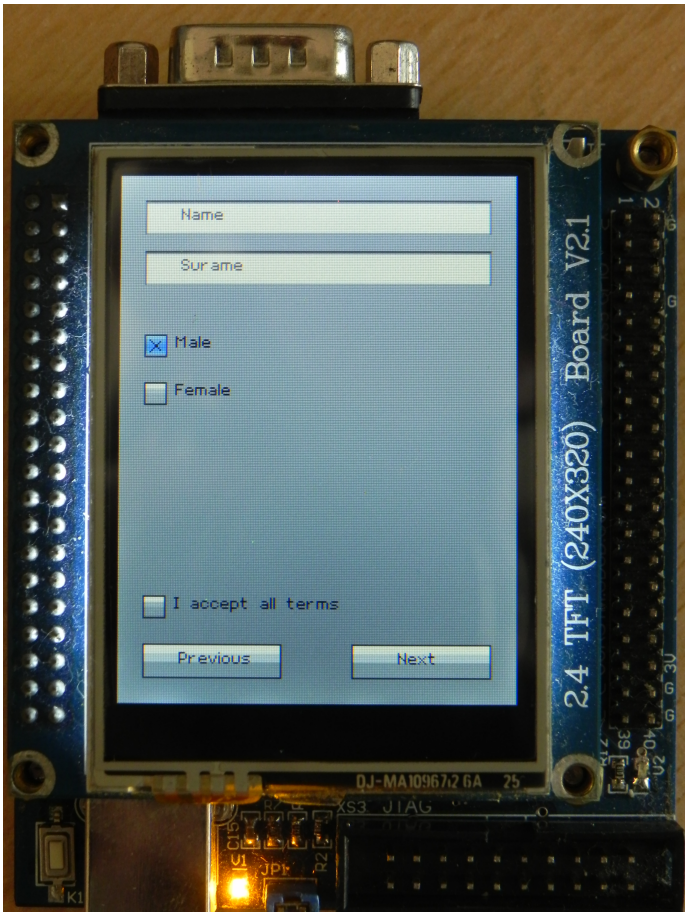
5.3 Porovnání výstupu se simulátorem

Při vývoji knihovny se simulátor opravdu ukázal jako užitečný nástroj, protože jsem měl možnost pracovat na OS Windows, Linux i Mac OS X a nemusel jsem v danou chvíli



Obrázek 9: Ukázková aplikace spuštěná v simulátoru

řešit, jak na dané platformě zprovoznit kompilátory pro ARM. Postupem času jsem si všímal určitých odchylek mezi simulací a běhu na vývojové sadě. Problém se ukázal být v koordinačním systému v Qt, který přičítá při kreslení obdélníků i tloušťku pera a to tak, že ji připočte pouze k jedné straně, což vedlo k problémům. Po opravě problému je výstup téměř identický, jediný rozdíl způsobuje písmo, které si každá platforma řeší individuálně. Porovnání je možné na obrázcích č.9 a 10.



Obrázek 10: Ukázková aplikace spuštěná ve vývojovém kitu

6 Závěr

Mým úkolem bylo naimplementovat knihovnu grafických prvků vhodnou pro architekturu ARM a ten se mi podařilo úspěšně splnit. Navíc jsem dodal i implementaci na platformě nezávislého simulátoru pro PC. Knihovna je navržena tak, aby se dala snadno rozšiřovat a aby měla jasnou architekturu. Dokáži si představit spoustu dalších funkcí, které je možné naimplementovat.

Například dynamická registrace služeb zpracovávající události, jelikož aktuální řešení vyžaduje předem zakompilovanou tabulku toho, komu a jak předávat události. Umožnilo by to rozšiřitelnost knihovny bez nutnosti rekompilace, a tak by byl programátor/uživatel schopný přidávat vlastní služby. Dále by bylo velice přínosné zavedení podpory barevných přechodů na základě výpočtů tak, aby je nebylo nutné zadávat barvu po barvě přímo do kódu.

Podpora automatického rozložení komponent je funkce, kterou by určitě ocenil snad každý programátor. Zároveň je to také velmi komplikovaná záležitost, protože je třeba, aby každý Widget uměl reportovat např. kolik místa minimálně potřebuje, a tím pádem se jedná o zásah do všech Widgetů a zavedení tříd řešících logiku nad nimi. Komplikovanost řešení by byla úměrná s mírou konfigurovatelnosti výsledného řešení.

Do knihovny by bylo také vhodné zařadit podporu pro klávesnici, ideálně dotekovou klávesnici na obrazovce, která by velice pomohla knihovně být všestrannější. Navíc je nyní možné veškerou novou funkcionalitu vyvíjet přímo na PC a pohodlně si ji ladit v Qt Creatoru, což velmi pomáhá rychlejšímu vyvíjení nových funkcí.

Pokud bych práci vypracovával znovu, určitě bych si dal více záležet na tom, abych připravil určitou sadu grafických operací jako referenci při psaní ovladače na jiné LCD řadiče. Celý proces by to určitě zrychlilo, protože by programátor nejdříve použil připravené algoritmy pro kreslení tvarů a jejich vybarvování a potom by je postupně odstraňoval a mohl by porovnávat výsledky.

Dále grafové komponenty nejsou úplně hotové, jelikož nevypisují konkrétní hodnoty u svých stupnic, a to by mohlo v určitých aplikacích už dělat někomu problém. Aktuálně jsou vhodné spíš pro orientační vizualizaci.

Bohužel jsem již neměl dostatek času pro podrobnou analýzu výkonu knihovny v testech, například za jak dlouho přepne 100 obrazovek, a tím určit minimální HW požadavky. Celou práci jsem psal také s důrazem na minimální využití operační paměti, ale přesně určit využití paměti jsem nestihl. Paměťová analýza navíc nebude úplně snadná, jelikož moje knihovna používá knihovnu stm32plus pro ARM a Qt pro PC, takže by bylo nutné připravit prázdnou implementaci na platformě závislé části a celou knihovnu zanalyzovat.

7 Reference

[1] Yui, Joseph, *The Definitive Guide to the ARM Cortex-M3* ISBN 1856179638